Sundials in the Shade
An Internet-Wide Perspective on ICMP Timestamps

Erik C. Rye(✉) and Robert Beverly

Naval Postgraduate School, Monterey, CA, USA
rye@cmand.org, rbeverly@nps.edu

Abstract. ICMP timestamp request and response packets have been standardized for nearly 40 years, but have no modern practical application, having been superseded by NTP. However, ICMP timestamps are not deprecated, suggesting that while hosts must support them, little attention is paid to their implementation and use. In this work, we perform active measurements and find 2.2 million hosts on the Internet responding to ICMP timestamp requests from over 42,500 unique autonomous systems. We develop a methodology to classify timestamp responses, and find 13 distinct classes of behavior. Not only do these behaviors enable a new fingerprinting vector, some behaviors leak important information about the host e.g., OS, kernel version, and local timezone.

Keywords: Network · Time · ICMP · Fingerprinting · Security

1 Introduction

The Internet Control Message Protocol (ICMP) is part of the original Internet Protocol specification (ICMP is IP protocol number one), and has remained largely unchanged since RFC 792 [21]. Its primary function is to communicate error and diagnostic information; well-known uses today include ICMP echo to test for reachability (i.e., `ping`), ICMP time exceeded to report packet loops (i.e., `traceroute`), and ICMP port unreachable to communicate helpful information to the initiator of a transport-layer connection. Today, 27 ICMP types are defined by the IESG, 13 of which are deprecated [11].

Among the non-deprecated ICMP messages are timestamp (type 13) and timestamp reply (type 14). These messages, originally envisioned to support time synchronization and provide one-way delay measurements [19], contain three 32-bit time values that represent milliseconds (ms) since midnight UTC. Modern clock synchronization is now performed using the Network Time Protocol [18] and ICMP timestamps are generally regarded as a potential security vulnerability [20] as they can leak information about a remote host's clock. Indeed, Kohno et al. demonstrated in 2005 the potential to identify individual hosts by variations in their clock skew [12], while [6] and [4] show similar discriminating power when fingerprinting wireless devices.

D. Choffnes and M. Barcellos (Eds.): PAM 2019, LNCS 11419, pp. 82–98, 2019.
https://doi.org/10.1007/978-3-030-15986-3_6

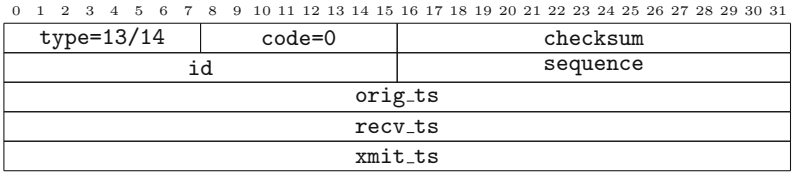

Fig. 1. ICMP timestamp message fields

In this work, we reassess the extent to which Internet hosts respond to ICMP timestamps. Despite no legitimate use for ICMP timestamps today, and best security practices that recommend blocking or disabling these timestamps, we receive timestamp responses from 2.2 million IPv4 hosts in 42,656 distinct autonomous systems (approximately 15% of the hosts queried) during a large-scale measurement campaign in September and October 2018. In addition to characterizing this unexpectedly large pool of responses, we seek to better understand *how* hosts respond. Rather than focusing on clock-skew fingerprinting, we instead make the following primary contributions:

1. The first Internet-wide survey of ICMP timestamp support and responsiveness.
2. A taxonomy of ICMP timestamp response behavior, and a methodology to classify responses.
3. Novel uses of ICMP timestamp responses, including fine-grained operating system fingerprinting and coarse geolocation.

2 Background and Related Work

Several TCP/IP protocols utilize timestamps, and significant prior work has examined TCP timestamps in the context of fingerprinting [12]. TCP timestamps have since been used to infer whether IPv4 and IPv6 server addresses map to the same physical machine in [2] and combined with clock skew to identify server "siblings" on a large scale in [24].

In contrast, this work focuses on ICMP timestamps. Although originally intended to support time synchronization [19], ICMP timestamps have no modern legitimate application use (having been superseded by NTP). Despite this, timestamps are not deprecated [11], suggesting that while hosts must support them, little attention is paid to their implementation and use.

Figure 1 depicts the structure of timestamp request (type 13) and response (type 14) ICMP messages. The 16-bit identifier and sequence values enable responses to be associated with requests. Three four-byte fields are defined: the *originate timestamp* (`orig_ts`), *receive timestamp* (`recv_ts`), and *transmit timestamp* (`xmit_ts`). Per RFC792 [21], timestamp fields encode milliseconds (ms) since UTC midnight unless the most significant bit is set, in which case the field may be a "non-standard" value. The originator of timestamp requests

should set the originate timestamp using her own clock; the value of the receive and transmit fields for timestamp requests is not specified in the RFC.

To respond to an ICMP timestamp request, a host simply copies the request packet, changes the ICMP type, and sets the receive and transmit time fields. The receive time indicates when the request was received, while the transmit time indicates when the reply was sent.

Several prior research works have explored ICMP timestamps, primarily for fault diagnosis and fingerprinting. Anagnostakis et al. found in 2003 that 93% of the approximately 400k routers they probed responded to ICMP timestamp requests, and developed a tomography technique using ICMP timestamps to measure per-link one-way network-internal delays [1]. Mahajan et al. leveraged and expanded the use of ICMP timestamps to enable user-level Internet fault and path diagnosis in [16].

Buchholz and Tjaden leveraged ICMP timestamps in the context of forensic reconstruction and correlation [3]. Similar to our results, they find a wide variety of clock behaviors. However, while they probe ∼8,000 web servers, we perform an Internet-wide survey including 2.2M hosts more than a decade later, and demonstrate novel fingerprinting and geolocation uses of ICMP timestamps.

Finally, the `nmap` security scanner [15] uses ICMP timestamp requests, in addition to other protocols, during host discovery for non-local networks in order to circumvent firewalls and blocking. `nmap` sets the request originate timestamp to zero by default, in violation of the standard [21] (though the user can manually specify a timestamp). Thus, ICMP timestamp requests with zero-valued origination times provide a signature of `nmap` scanners searching for live hosts. While `nmap` uses ICMP timestamps for liveness testing, it does not use them for operating system detection as we do in this work.

To better understand the prevalence of ICMP timestamp scanners, we analyze 240 days of traffic arriving at a /17 network telescope. We observe a total of 413,352 timestamp messages, 93% of which are timestamp requests. Only 33 requests contain a non-zero originate timestamp, suggesting that the remainder (nearly 100%) are `nmap` scanners. The top 10 sources account for more than 86% of the requests we observe, indicating a relatively small number of active Internet-wide scanners.

3 Behavioral Taxonomy

During initial probing, we found significant variety in timestamp responses. Not only do structural differences exist in the implementation of [21] by timestamp-responsive routers and end systems (e.g., little- vs big-endian), they also occur relative to how the device counts time (e.g., milliseconds vs. seconds), the device's reference point (e.g., UTC or local time), whether the reply is a function of request parameters, and even whether the device is keeping time at all.

Table 1. ICMP timestamp classification fingerprints

Num	Class	Request	Response		
		cksum	orig_ts	recv_ts	xmit_ts
1	Normal	Valid	-	$\neq$ xmit_ts, $\neq 0$	$\neq 0$
2	Lazy	Valid	-	= xmit_ts	$\neq 0$
3	Checksum-Lazy	Bad	-	-	-
4	Stuck	valid	-	const	const
5	Constant 0	Valid	-	0	0
6	Constant 1	Valid	-	1	1
7	Constant LE 1	Valid	-	$htonl(1)$	$htonl(1)$
8	Reflection	Valid	-	$\texttt{request}_{\texttt{recv\_ts}}$	$\texttt{request}_{\texttt{xmit\_ts}}$
9	Non-UTC	Valid	-	$>2^{31}-1$	$>2^{31}-1$
10	Timezone	Valid	-	$\lvert\texttt{recv\_ts} - \texttt{orig\_ts}\rvert \% \left(3.6 \times 10^6\right) < 200\,\text{ms}$	-
11	Little Endian	Valid	-	$\lvert htonl(\texttt{recv\_ts}) - \texttt{orig\_ts}\rvert < 200\,\text{ms}$	-
12	Linux *htons*() Bug	Valid	-	$\%2^{16} = 0$	$\%2^{16} = 0$
13	Unknown	Valid	-	-	-

3.1 Timestamp Implementation Taxonomy

Table 1 provides an exhaustive taxonomy of the behaviors we observe; we term these the ICMP timestamp *classifications*. Note that this taxonomy concerns only the *implementation* of the timestamp response, rather than whether the responding host's timestamp values are correct.

- **Normal:** Conformant to [21]. Assuming more than one ms of processing time, the receive and transmit timestamps should be not equal, and both should be nonzero except at midnight UTC.
- **Lazy:** Performs a single time lookup and sets both receive and transmit timestamp fields to the same value. A review of current Linux and FreeBSD kernel source code reveals this common lazy implementation [10,13].
- **Checksum-Lazy:** Responds to timestamp requests even when the ICMP checksum is incorrect.
- **Stuck:** Returns the same value in the receive and transmit timestamp fields regardless of the input sent to it and time elapsed between probes.
- **Constant 0, 1, Little-Endian 1:** A strict subset of "stuck" that always returns a small constant value in the receive and transmit timestamp fields.
- **Reflection:** Copies the receive and transmit timestamp fields from the timestamp request into the corresponding fields of the reply message[1].
- **Non-UTC:** Receive and transmit timestamp values with the most significant bit set. As indicated in [21], network devices that are unable to provide a timestamp with respect to UTC midnight or in ms may use an alternate time source, provided that the high order bit is set.
- **Linux *htons()* Bug:** Certain versions of the Linux kernel (and Android) contain a flawed ICMP timestamp implementation where replies are truncated to a 16-bit value; see Appendix A for details.
- **Unknown:** Any reply not otherwise classified.

[1] We find no copying of originate timestamp into the reply's receive or transmit fields.

3.2 Timekeeping Behavior Taxonomy

We next categorize the types of timestamp responses we observe by what the host is measuring and what they are measuring in relation to.

- **Precision:** Timestamp reply fields should encode ms to be conformant, however some implementations encode seconds.
- **UTC reference:** Conformant to the RFC; receive and transmit timestamps encode ms since midnight UTC.
- **Timezone:** Replies with receive and transmit timestamps in ms relative to midnight in the device's local timezone, rather than UTC midnight.
- **Epoch reference:** Returned timestamps encode time in seconds relative to the Unix epoch time.
- **Little-Endian:** Receive and transmit timestamps containing a correct timestamp when viewed as little-endian four-byte integers.

4 Methodology

We develop `sundial`, a packet prober that implements the methodology described herein to elicit timestamp responses that permit behavioral classification. `sundial` is written in C and sends raw IP packets in order to set specific IP and ICMP header fields, while targets are randomized to distribute load. We have since ported `sundial` to a publicly available ZMap [8] module [22].

Our measurement survey consists of probing 14.5 million IPv4 addresses[2] of the August 7, 2018 ISI hitlist, which includes one address per routable /24 network [9]. We utilize two vantage points connected to large academic university networks named after their respective locations: "Boston" and "San Diego." Using `sundial`, we elicit ICMP timestamp replies from ∼2.2 million unique IPs.

This section first describes `sundial`'s messages and methodology, then our ground truth validation. We then discuss ethical concerns and precautions undertaken in this study.

4.1 `sundial` Messages

In order to generate and categorize each of the response behaviors, `sundial` transmits four distinct types of ICMP timestamp requests. Both of our vantage points have their time NTP-synchronized to stratum 2 or better servers. Thus time is "correct" on our prober relative to NTP error.

1. **Standard:** We fill the originate timestamp field with the correct ms from UTC midnight, zero the receive and transmit timestamp fields, and place the lower 32 bits of the MD5 hash of the destination IP address and originate timestamp into the identifier and sequence number fields. The hash permits detection of destinations or middleboxes that tamper with the originate timestamp, identifier, or sequence number.

[2] As IPv6 does not support timestamps in ICMPv6, we study IPv4 exclusively.

2. **Bad Clock:** We zero the receive and transmit fields of the request, choose an identifier and sequence number, and compute the MD5 hash of the destination IP address together with the identifier and sequence number. The lower 32 bits of the hash are placed in the originate timestamp. This hash again provides the capability to detect modification of the reply.
3. **Bad Checksum:** The correct time in ms since UTC midnight is placed in the originate field, the receive and transmit timestamps are set to zero, and the identifier and sequence number fields contain an encoding of the destination IP address along with the originate timestamp. We deliberately choose a random, incorrect checksum and place it into the ICMP timestamp request's checksum field. This timestamp message should appear corrupted to the destination, and a correct ICMP implementation should discard it.
4. **Duplicate Timestamp:** The receive and transmit timestamps are initialized to the originate timestamp value by the sender, setting all three timestamps to the same correct value. The destination IP address and originate timestamp are again encoded in the identifier and sequence number to detect modifications.

Many implementation behaviors in Sect. 3 can be inferred from the first, standard probe. For instance, the standard timestamp request can determine a normal, lazy, non-UTC and little-endian implementation. In order to classify a device as stuck, both the standard and duplicate timestamp requests are required. Two requests are needed in order to determine that the receive and transmit timestamps remain fixed over time, and the inclusion of the duplicate timestamp request ensures that the remote device is not simply echoing the values in the receive and transmit timestamp fields of the request. Similarly, timestamp reflectors can be detected using the standard and duplicate request responses.

The checksum-lazy behavior is detected via responses to the bad checksum request type. The Linux `htons()` bug behavior can be detected using the standard request and filtering for reply timestamps with the two lower bytes set to zero. In order to minimize the chance of false positives (i.e., the correct time in ms from UTC midnight is represented with the two lower bytes zeroed), we count only destinations that match this behavior in responses from both the standard and bad clock timestamp request types.

To detect the unit precision of the timestamp reply fields, we leverage the multiple requests sent to each target. Because we know the time at which requests are transmitted, we compare the time difference between the successive requests to a host and classify them based on the inferred time difference from the replies.

Finally, we classify responsive devices by the reference by which they maintain time. We find many remote machines that observe nonstandard reference times, but do not set the high order timestamp field bit. A common alternative timekeeping methodology is to track the number of ms elapsed since midnight local time. We detect local timezone timekeepers by comparing the receive and transmit timestamps to the originate timestamp in replies to the standard request. Receive and transmit timestamps that differ from our correct originate

Table 2. Ground truth classification of ICMP timestamp behaviors

OS	Behavior	Notes
Windows 7–10	Off by default	With Windows firewall off, lazy LE
Linux	Lazy	
Linux 3.18 (incl Android)	Lazy	htons() bug
Android kernel 3.10, 4.4+	Lazy	
BSD	Lazy	
OSX	Unresponsive	
iOS	Off by default	
Cisco IOS/IOS-XE	Lazy	MSB set if NTP disabled, unset if enabled
JunOS	Lazy	

timestamp by the number of ms for an existing timezone (within an allowable error discussed in Sect. 5.2) are determined to be keeping track of their local time.

Last, a small number of devices we encountered measured time relative to the Unix epoch. Epoch-relative timestamps are detected in two steps: first, we compare the epoch timestamp's date to the date in which we sent the request; if they match, we determine whether the number of seconds elapsed since UTC midnight in the reply is suitably close to the correct UTC time.

4.2 Ground Truth

To validate our inferences and understand the more general behavior of popular operating systems and devices, we run `sundial` against a variety of known systems; Table 2 lists their ICMP timestamp reply behavior.

Apple desktop and mobile operating systems, macOS and iOS, both do not respond to ICMP timestamp messages by default. Initially, we could not elicit any response from Microsoft Windows devices, until we disabled Windows Firewall. Once disabled, the Windows device responds with correct timestamps in little-endian byte order. This suggests that not only are timestamp-responsive devices with little-endian timestamp replies Windows, but it also worryingly indicates that its built-in firewall has been turned off by the administrator.

BSD and Linux devices respond with lazy timestamp replies, as their source code indicates they should. JunOS and Android respond like FreeBSD and Linux, on which they are based, respectively. Of note, we built the Linux 3.18 kernel, which has the `htons()` bug described in Sect. 6; it responded with the lower two bytes zeroed, as expected. This bug has made its way into Android, where we find devices running the 3.18 kernel exhibiting the same signature.

Cisco devices respond differently depending on whether they have enabled NTP. NTP is not enabled by default on IOS; the administrator must manually enable the protocol and configure the NTP servers to use. If NTP has not been enabled, we observe devices setting the most significant bit, presumably

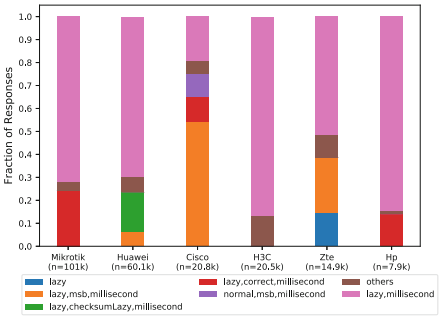

Fig. 2. Incidence of fingerprints for most common telnet banner manufacturers

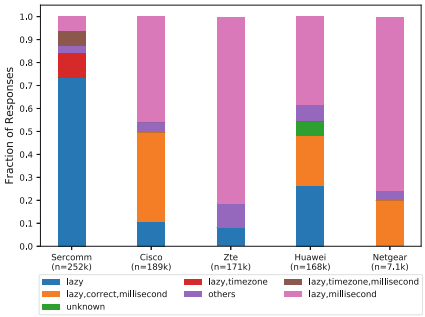

Fig. 3. Incidence of fingerprints for most common CWMP scan manufacturers

to indicate that it is unsure whether the timestamp is accurate, and filling in a UTC-based timestamp with the remaining bits, according to its internal clock.

Telnet Banner and CWMP GET Ground Truth. To augment the ground truth we obtained from devices we were able to procure locally, we leveraged IPv4 Internet-wide Telnet banner- and CPE WAN Management Protocol (CWMP) parameter-grabbing scans from `scans.io` [23]. From October 3, 2018 scans, we search banners (Telnet) and GET requests (CWMP) for IP addresses associated with known manufacturer strings. We then probe these addresses with `sundial`.

Figure 2 displays the most common fingerprints for a subset of the manufacturers probed from `scans.io`'s Telnet banner-grab dataset, while Fig. 3 is the analogous CWMP plot. We note that non-homogeneous behavior within a manufacturer's plot may be due to several factors: different behaviors among devices of the same manufacturer, banner spoofing, IP address changes, and middleboxes between the source and destination. We provide further details regarding our use of the `scans.io` datasets in Appendix B.

4.3 Ethical Considerations

Internet-wide probing invariably raises ethical concerns. We therefore follow the recommended guidelines for good Internet citizenship provided in [8] to mitigate the potential impact of our probing. At a high-level, we only send ICMP packets, which are generally considered less abusive than e.g., TCP or UDP probes that may reach active application services. Further, our pseudo-random probing order is designed to distribute probes among networks in time so that they do not appear as attack traffic. Finally, we make an informative web page accessible via the IP address of our prober, along with instructions for opting-out. In this work, we did not receive any abuse reports or opt-out requests.

5 Results

On October 6, 2018, we sent four ICMP timestamp request messages as described in Sect. 4.1 from both of our vantage points to each of the 14.5 million target IPv4 addresses in the ISI hitlist. We obtained at least one ICMP timestamp reply message from 2,221,021 unique IP addresses in 42,656 distinct autonomous systems as mapped by Team Cymru's IP-to-ASN lookup service [5]. Our probing results are publicly available [22].

We classify the responses according to the implementation taxonomy outlined in Sect. 3 and Table 1, the timekeeping behavior detailed in Sect. 3.2, and the correctness of the timestamp reply according to Sect. 5.2. Tables 3 and 4 summarize our results in tabular form; note that the implementation behavior categories are not mutually exclusive, and the individual columns will sum to more than the total column, which is the number of unique responding IP addresses. We received replies from approximately 11,000 IP addresses whose computed MD5 hashes as described in Sect. 4.1 indicated tampering of the source IP address, originate timestamp, or id and sequence number fields; we discard these replies.

5.1 Macro Behavior

Lazy replies outnumber normal timestamp replies by a margin of over 50 to 1. Because we had assumed the normal reply type would be the most common, we investigated open-source operating systems' implementations of ICMP. In both the Linux and BSD implementations, the receive timestamp is filled in via a call to retrieve the current kernel time, after which this value is simply copied into the transmit timestamp field. Therefore, all BSD and Linux systems, and their derivatives, exhibit the lazy timestamp reply behavior.

Normal hosts can appear lazy if the receive and transmit timestamps are set within the same millisecond. This ambiguity can be resolved in part via multiple probes. For instance, Table 3 shows that only ∼50% of responders classified as normal by one vantage are also marked normal by the other.

The majority (61%) of responding devices do not reply with timestamps within 200 ms of our NTP-synchronized reference clock, our empirically-derived correctness bound discussed in Sect. 5.2. Only ∼40% of responding IP addresses fall into this category; notably, we detect smaller numbers devices with correct clocks incorrectly implementing the timestamp reply message standard. For example, across both vantage points we detect thousands of devices whose timestamps are correct when interpreted as a little-endian integer, rather than in network byte order. We discover one operating system that implements little-endian timestamps in Sect. 4.2. In another incorrect behavior that nevertheless indicates a correct clock, some devices respond with the correct timestamp and the most significant bit set – a behavior at odds with the specification [21] where the most significant bit indicates a timestamp either not in ms, or the host cannot provide a timestamp referenced to UTC midnight. In Sect. 4.2, we discuss an operating system that sets the most significant bit when its clock has not been synchronized with NTP.

Table 3. Timestamp reply implementation behaviors (values do not sum to total)

Category	Boston	Both	San Diego	Category	Boston	Both	San Diego
Normal	40,491	19,819	40,363	Stuck	855	849	873
Lazy	2,111,344	1,899,297	2,112,386	Constant 0	547	546	555
Checksum-Lazy	28,074	23,365	28,805	Constant 1	200	199	207
Non-UTC	249,454	211,755	249,932	Constant LE 1	22	19	23
Reflection	2,325	2,304	2,364	`htons()` Bug	1,499	665	1,536
Correct	850,787	803,314	850,133	Timezone	33,317	23,464	33,762
Correct LE	11,127	5,244	11,290	Unknown	38,495	11,865	32,956
Correct - MSB	1,048	386	973				
Total					2,194,180	1,934,172	2,189,524

Over 200,000 unique IPs (>10% of each vantage point's total) respond with the most significant bit set in the receive and transmit timestamps; those timestamps that are otherwise correct are but a small population of those we term Non-UTC due to the prescribed meaning of this bit in [21]. Some hosts and routers fall into this category due to the nature of their timestamp reply implementation – devices that mark the receive and transmit timestamps with little-endian timestamps will be classified as Non-UTC if the most significant bit of the lowest order byte is on, when the timestamp is viewed in network byte order. Others, as described above, turn on the Non-UTC bit if they have not synchronized with NTP.

Another major category of non-standard implementation behavior of ICMP timestamp replies are devices that report their timestamp relative to their local timezone. Whether devices are programmatically reporting their local time without human intervention, or whether administrator action is required to change the system time (from UTC to local time) in order to effect this classification is unclear. In either case, timezone timestamp replies allow us to coarsely geolocate the responding device. We delve deeper into this possibility in Sect. 5.4.

Finally, while most responding IP addresses are unsurprisingly classified as using milliseconds as their unit of measure, approximately 14–16% of IP addresses are not (see Table 4). In order to determine what units are being used in the timestamp, we subtract the time elapsed between the standard timestamp request and duplicate timestamp request, both of which contain correct originate timestamp fields. We then subtract the time elapsed according to the receive and transmit timestamps in the timestamp reply messages. If the difference of differences is less than 400 ms (two times 200 ms, the error margin for one reply) we conclude that the remote IP is counting in milliseconds. A similar calculation is done to find devices counting in seconds. Several of the behavioral categories outlined in Sect. 3.1 are included among the hosts with undefined timekeeping behavior – those whose clocks are stuck at a particular value and those that reflect the request's receive and transmit timestamps into the corresponding fields are two examples. Others may be filling the reply timestamps with random values.

Table 4. Timestamp reply timekeeping behaviors

Category	Boston	Both	San Diego
Millisecond	1,826,696	1,722,176	1,866,529
Second	47	37	68
Epoch	1	1	1
Unknown timekeeping	367,436	211,958	322,926
Total	2,194,180	1,934,172	2,189,524

5.2 Timestamp Correctness

In order to make a final classification – whether the remote host's clock is correct or incorrect – as well as to assist in making many of the classifications within our implementation and timekeeping taxonomies that require a correctness determination, we describe in this section our methodology for determining whether or not a receive or transmit timestamp is correct.

To account for clock drift and network delays, we aim to establish a margin of error relative to a correctly marked originate timestamp, and consider receive and transmit timestamps within that margin from the originate timestamp to be correct. To that end, we plot the probability density of the differences between the receive and originate timestamps from 2.2 million timestamp replies generated by sending a single standard timestamp request to each of 14.5 million IP addresses from the ISI hitlist [9] in Fig. 4.

Figure 4 clearly depicts a trough in the difference probability values around 200 ms, indicating that receive timestamps greater than 200 ms than the originate timestamp are less likely than those between zero and 200 ms. We reflect this margin about the y-axis, despite the trough occurring somewhat closer to the origin on the negative side. Therefore, we declare a timestamp correct if it is within our error margin of 200 ms of the originate timestamp.

5.3 Middlebox Influence

To investigate the origin of some of the behaviors observed in Sect. 3 for which we have no ground truth implementations, we use `tracebox` [7] to detect middleboxes. In particular, we chose for investigation hosts implementing the reflection, lazy with MSB set (but not counting milliseconds), and constant 0 behaviors, as we do not observe any of these fingerprints in our ground truth dataset, yet there exist nontrivial numbers of them in our Internet-wide dataset.

In order to determine whether a middlebox may be responsible for these behaviors for which we have no ground truth, we `tracebox` to a subset of 500 random IP addresses exhibiting them. For our purposes, we consider an IP address to be behind a middlebox if the last hop modifies fields beyond the standard IP TTL and checksum modifications, and DSCP and MPLS field alterations and extensions. Of 500 reflection IP addresses, only 44 showed evidence of being

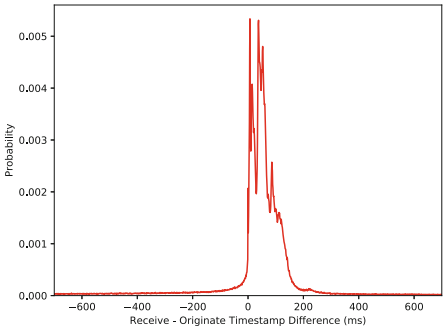

Fig. 4. Empirical `recv_ts- orig_ts` PMF

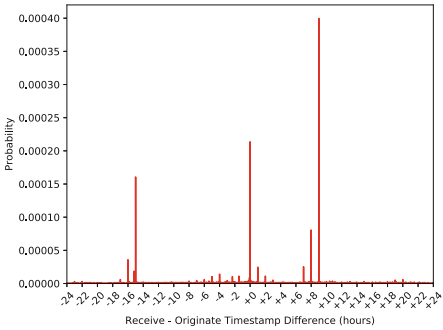

Fig. 5. Response error; note hourly peaks

behind a middlebox, suggesting that some operating systems implement the reflect behavior and that this is a less common middlebox modification. The lazy with MSB set (but non-ms counting) behavior, on the other hand, was inferred to be behind a middlebox in 333 out of 500 random IP addresses, suggesting it is most often middleboxes that are causing the lazy-MSB-set fingerprint. Finally, about half of the constant 0 IP addresses show middlebox tampering in `tracebox` runs, suggesting that this behavior is both an operating system implementation of timestamp replies as well as a middlebox modification scheme.

5.4 Geolocation

Figure 5 displays the probability distribution of response error, e.g., `recv_ts` − `orig_ts`, after correct replies have been removed from the set of standard request type responses. While there is a level of uniform randomness, we note the peaks at hour intervals. We surmise that these represent hosts that have correct time, but return a *timezone-relative* response (in violation of the standard [21] where responses should be relative to UTC). The origin of timezone-relative responses may be a non-conformant implementation. Alternatively, these responses may simply be an artifact of non-NTP synchronized machines where the administrator instead sets the localtime correctly, but incorrectly sets the timezone. In this case, the machine's notion of UTC is incorrect, but incorrect relative to the set timezone. Nevertheless, these timezone-relative responses effectively leak the host's timezone. We note the large spike in the +9 timezone, which covers Japan and South Korea; despite the use of `nmap`'s OS-detection feature, and examining web pages and TLS certificates where available, we could not definitively identify a specific device manufacturer or policy underpinning this effect.

To evaluate our ability to coarsely geolocate IP addresses reporting a timezone-relative timestamp, we begin with ∼34,000 IP addresses in this category obtained by sending a single probe to every hitlist IP from our Boston vantage. Using the reply timestamps, we compute the remote host's local timezone

offset relative to UTC to infer the host's timezone. We then compare our inferred timezone with the timezone reported by the MaxMind GeoLite-2 database [17].

For each IP address, we compare the MaxMind timezone's standard time UTC-offset and, if applicable, daylight saving time UTC offset, to the timestamp-inferred offset. Of the 34,357 IP addresses tested, 32,085 (93%) correctly matched either the standard timezone UTC offset or daylight saving UTC offset, if the MaxMind-derived timezone observes daylight saving time. More specifically, 18,343 IP addresses had timestamp-inferred timezone offsets that matched their MaxMind-derived timezone, which did not observe daylight saving time. 11,188 IP addresses resolved to a MaxMind timezone, whose daylight saving time offset matched the offset inferred from the timestamp. 2,554 IP addresses had timestamp-inferred UTC offsets that matched their MaxMind-derived standard time offset for timezones that do observe daylight saving time. Of the inferred UTC-offsets that were not correct, 1,641 did not match either the standard time offset derived from MaxMind, or the daylight saving time offset, if it existed, and 631 IP addresses did not resolve to a timezone in MaxMind's free database.

6 Conclusions and Future Work

We observe a wide variety of implementation behavior of the ICMP timestamp reply type, caused by timestamps' lack of a modern use but continued requirement to be supported. In particular, we are able to uniquely fingerprint the behavior of several major operating systems and kernel versions, and geolocate Internet hosts to timezone accuracy with >90% success.

As future work, we intend to exhaustively scan and classify the IPv4 Internet, scan a subset with increased frequency over a sustained time period, and to do so many vantage points. We further plan to integrate the OS-detection capabilities we uncover in this work into `nmap`, and add `tracebox` functionality to `sundial` in order to better detect middlebox tampering with ICMP timestamp messages.

Acknowledgments. We thank Garrett Wollman, Ram Durairajan, and Dan Andersen for measurement infrastructure, our shepherd Rama Padmanabhan, and the anonymous reviewers for insightful feedback. Views and conclusions are those of the authors and not necessarily those of the U.S. government.

Appendix A: Linux `htons()` Bug

While investigating the source code of open-source operating systems' implementation of ICMP timestamps, we observed a flaw that allows fine-grained fingerprinting of the Linux kernel version 3.18. The specific bug that allows this fingerprinting was introduced in March 2016. An update to the Internet timestamp generating method in `af_inet.c` errantly truncated the 32-bit timestamp to a 16-bit short via a call to the C library function `htons()` rather than `htonl()`. When this incorrect 16-bit value is placed into the 32-bit receive and transmit timestamp fields of a timestamp reply, it causes the lower two bytes

to be zero and disables the responding machine's ability to generate a correct reply timestamp at any time other than midnight UTC. This presents a unique signature of devices running the Linux kernel built during this time period. In order to identify these devices on the Internet, we filter for ICMP timestamp replies containing receive and transmit timestamp values with zeros in the lower two bytes when viewed as a 32-bit big-endian integer. While devices that are correctly implementing ICMP timestamp replies will naturally reply with timestamps containing zeros in the lower two bytes every 65,536 milliseconds, the probability of multiple responses containing this signature drops rapidly as the number of probes sent increases.

Being derived directly from the Linux kernel, the 3.18 version of the Android kernel also includes the flawed `af_inet.c` implementation containing the same `htons()` truncation, allowing for ICMP timestamp fingerprinting of mobile devices as well.

While Linux 3.18 reached its end of life [14] in 2017, we observe hosts on the Internet whose signatures suggest this is the precise version of software they are currently running. Unfortunately, this presents an adversary with the opportunity to perform targeted attacks.

Appendix B: `scans.io` Ground Truth

We use Telnet and CWMP banners in public `scans.io` as a source of ground truth. It is possible to override the default text of these protocol banners, and recognize that this is a potential source of error. However, we examine the manufacturer counts in aggregate under the assumption that most manufacturer strings are legitimate. We believe it unlikely that users have modified their CWMP configuration on their customer premises equipment to return an incorrect manufacturer.

Parsing the Telnet and CWMP scans for strings containing the names of major network device manufacturers provided over two million unique IP addresses. Table 5 summarizes the results; note that for some manufacturers (e.g., Arris) approximately the same number of IPs were discovered through the Telnet scan as the CWMP scan, for others (e.g., Cisco and Huawei) CWMP provided an order of magnitude greater number of IPs, and still others (e.g., Mikrotik and Netgear) appeared in only one of the two protocol scans. Note that these numbers are not the number of timestamp-responsive IP addresses denoted by n in Figs. 2 and 3.

With the IP addresses we obtained for each manufacturer, we then run `sundial` to each set in order to elicit timestamp reply fingerprints and determine whether different manufacturers tend to exhibit unique reply behaviors. Figures 2 and 3 display the incidence of timestamp reply fingerprints for a subset of the manufacturers we probed, and provide some interesting results that we examine here in greater detail.

No manufacturer exhibits only a singular behavior. We attribute this variety within manufacturers to changes in their implementation of timestamp replies

Table 5. Unique IP addresses per manufacturer for each scan

Manufacturer	Telnet count	CWMP count
Arris	8,638	5,281
Cisco	29,135	1,298,761
H3C	80,445	-
HP	24,027	-
Huawei	170,710	2,377,079
Mikrotik	190,484	-
Netgear	-	17,723
Sercomm	-	899,492
Ubiquiti	598	-
Zhone	6,999	-
ZTE	17,972	560,177
Zyxel	5,902	-

over time, different implementations among different development or product groups working with different code bases, and the incorporation of outside implementations inherited through acquisitions and mergers.

Second, we are able to distinguish broad outlines of different manufacturers based on the incidence of reply fingerprints. In Fig. 2, we note that among the top six manufacturers, only Huawei had a significant number of associated IP addresses (∼10%) that responded with the checksum-lazy behavior. More than half of the Cisco IP addresses from the Telnet scan exhibited the lazy behavior with the most significant bit set while counting milliseconds, a far greater proportion than any other manufacturer. Also noteworthy is that none of the manufacturers represented in the Telnet scan exhibits large numbers of correct replies. In our Telnet data, Mikrotik devices responded with a correct timestamp reply roughly 25% of the time, a higher incidence than any other manufacturer. This suggests that perhaps certain Mikrotik products have NTP enabled by default, allowing these devices to obtain correct time more readily than those that require administrator interaction. Our CWMP results in Fig. 3 demonstrate the ability to distinguish manufacturer behavior in certain cases as well, we note the >70% of Sercomm devices that exhibit only the lazy behavior, as well as Sercomm exhibiting the only timezone-relative timekeeping behavior among the CWMP manufacturers.

Finally, we note differences between the protocol scans among IP addresses that belong to the same manufacturer. Cisco, Huawei, and ZTE appear in both protocol results in appreciable numbers, and are represented in both figures in Sect. 4.2. Although Cisco devices obtained from the Telnet scan infrequently (∼10%) respond with correct timestamps, in the CWMP data the proportion is nearly 40%. Huawei devices from the Telnet data are generally lazy responders that count in milliseconds, however, this same behavior occurs only half as

frequently in the CWMP data. Further, the fingerprint consisting solely of the lazy behavior represents nearly a quarter of the CWMP Huawei devices, while it is insignificant in the Telnet Huawei data. While the differences between the Telnet and CWMP data are less pronounced for ZTE, they exist as well in the lack of appreciable numbers of ZTE devices setting the most significant bit in replies within the CWMP corpus.

Appendix C: Timezone-Relative Behavior

Figure 5 displays the probability mass function of the differences between the receive and originate timestamps for a `sundial` scan conducted on 9 September 2018 from the Boston vantage after responses with correct timestamps have been removed. Discernible peaks occur at many of the hourly intervals representing timezone-relative responders, rising above a base level of randomness. The hourly offsets in Fig. 5 may need to be normalized to the range of UTC timezone offsets, however. For example, depending on the originate timestamp value, a responding host's receive timestamp at a UTC offset of +9 may appear either nine hours ahead of the originate timestamp, or 15 h behind, as $-15 \equiv 9 (\mathrm{mod}\ 24)$. In Fig. 5 we see large spikes at both +9 and −15 h, but in reality these spikes represent the same timezone.

Table 6. Inferred UTC-offsets from timestamp replies

UTC offset	−12	−11	−10	−9	−8	−7	−6	−5	−4	−3.5	−3	−2	−1	1	2
Count	73	1	7	3	386	476	666	1,763	2,660	2	246	228	5	7,215	1,819
UTC offset	3	3.5	4	4.5	5	5.5	6	6.5	7	8	9	9.5	10	11	
Count	449	8	62	3	87	17	14	13	565	3,496	13,861	6	215	11	

We identify timezone-relative responses systematically by computing the local time in milliseconds for each of the UTC-offsets detailed in Table 6, given the originate timestamp contained in the timestamp response. We then compare each candidate local timezone's originate timestamp to the receive timestamp in the reply. If the candidate originate timestamp is within the 200 ms correctness bound established in Sect. 5.2, we classify the IP address as belonging to the timezone that produced the correct originate timestamp. Table 6 details the number of timezone-relative responders we identified during the 9 September `sundial` scan.

References

1. Anagnostakis, K.G., Greenwald, M., Ryger, R.S.: cing: Measuring network-internal delays using only existing infrastructure. In: Twenty-Second Annual Joint Conference of the IEEE Computer and Communications, vol. 3, pp. 2112–2121 (2003)

2. Beverly, R., Berger, A.: Server siblings: identifying shared IPv4/IPv6 infrastructure via active fingerprinting. In: Mirkovic, J., Liu, Y. (eds.) PAM 2015. LNCS, vol. 8995, pp. 149–161. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15509-8_12
3. Buchholz, F., Tjaden, B.: A brief study of time. Digit. Invest. **4**, 31–42 (2007)
4. Cristea, M., Groza, B.: Fingerprinting smartphones remotely via ICMP timestamps. IEEE Commun. Lett. **17**(6), 1081–1083 (2013)
5. Cymru, Team: IP to ASN mapping (2008). https://www.team-cymru.org/IP-ASN-mapping.html
6. Desmond, L.C.C., Yuan, C.C., Pheng, T.C., Lee, R.S.: Identifying unique devices through wireless fingerprinting. In: Proceedings of the First ACM Conference on Wireless Network Security, pp. 46–55 (2008)
7. Detal, G., Hesmans, B., Bonaventure, O., Vanaubel, Y., Donnet, B.: Revealing middlebox interference with tracebox. In: ACM SIGCOMM Internet Measurement Conference, pp. 1–8 (2013)
8. Durumeric, Z., Wustrow, E., Halderman, J.A.: ZMap: fast internet-wide scanning and its security applications. In: USENIX Security, pp. 605–620 (2013)
9. Fan, X., Heidemann, J.: Selecting representative IP addresses for Internet topology studies. In: ACM SIGCOMM Internet Measurement Conference, pp. 411–423 (2010)
10. FreeBSD: FreeBSD Kernel ICMP Code, SVN Head (2018). https://svnweb.freebsd.org/base/head/sys/netinet/ip_icmp.c?revision=336677
11. Internet Engineering Standards Group: Internet Control Message Protocol (ICMP) Parameters (2018). https://www.iana.org/assignments/icmp-parameters/icmp-parameters.xhtml
12. Kohno, T., Broido, A., Claffy, K.C.: Remote physical device fingerprinting. IEEE Trans. Dependable Secure Comput. **2**(2), 93–108 (2005)
13. Linux: Linux Kernel ICMP Code, Git Head (2018). https://github.com/torvalds/linux/blob/master/net/ipv4/icmp.c
14. Linux: The Linux Kernel Archives (2018). https://www.kernel.org/
15. Lyon, G.: Nmap Security Scanner. https://nmap.org
16. Mahajan, R., Spring, N., Wetherall, D., Anderson, T.: User-level internet path diagnosis. ACM SIGOPS Oper. Syst. Rev. **37**(5), 106–119 (2003)
17. MaxMind: GeoLite2 IP Geolocation Databases (2018). https://dev.maxmind.com/geoip/geoip2/geolite2/
18. Mills, D., Martin, J., Burbank, J., Kasch, W.: Network Time Protocol Version 4: Protocol and Algorithms Specification. RFC 5905 (Proposed Standard), June 2010. http://www.ietf.org/rfc/rfc5905.txt
19. Mills, D.: DCNET Internet Clock Service. RFC 778 (Historic), April 1981. http://www.ietf.org/rfc/rfc778.txt
20. MITRE: CVE-1999-0524. Available from MITRE, CVE-ID CVE-1999-0524, August 1999. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0524
21. Postel, J.: Internet Control Message Protocol. RFC 792 (INTERNET STANDARD), September 1981. http://www.ietf.org/rfc/rfc792.txt
22. Rye, E.C.: Sundial ICMP Timestamp Inference Tool (2019). https://www.cmand.org/sundial
23. Scans.io: Internet-Wide Scan Data Repository. https://scans.io
24. Scheitle, Q., Gasser, O., Rouhi, M., Carle, G.: Large-scale classification of IPv6-IPv4 siblings with variable clock skew. In: 2017 Network Traffic Measurement and Analysis Conference (TMA), pp. 1–9. IEEE (2017)